Süsteemprogrammeerimine keeles C

Loeng 12

chroot()

- Vahetab juurkataloogi etteantud kataloogiks

```
#include <unistd.h>
int chroot (const char *path)
```

- Kasutatakse programmide vangistamiseks etteantud kataloogi
- Käivitada tohib ainult juurkasutaja (tegelikult CAP_SYS_CHROOT protsesside puhul)
- Süsteemi juurkataloog muudetakse protsessi jaoks ära
- / algav *path* hakkab näitama vastavasse kataloogi nagu ka antud kataloogi .. (**NB! Ei midagi rohkem**)

chroot() kasutamine

- Mida jälgida?
  - chroot() nõuab protsessilt vastavat privileegi
  - pärast chrooti tuleb jälgida, et protsessil poleks väljaspool "vanglasüsteemi" avatud faile (fchdir käsu oht)
  - töökataloog tuleb muuta chrooditavaks kataloogiks ise
  - hea oleks ka kasutaja setuid()-ga ära muuta (et väljamurdmiseks uuesti chroot'i ei saaks teha)
  - teiste programmide ja librade antud keskkonnas kasutamiseks on vaja nad sellesse sisse tuua

Mäluhaldusest uuesti

- Reeglina saab protsess suure virtuaalse aadressruumi (aadress 0 kuni aadress "vägasuur")
- Ei pruugi olla jätkuv: igale aadressile ei saa andmeid salvestada
- Jaguneb *page*ideks. (tüüpiliselt mahuga 4kB)
- Iga *page* elab kas päris mälus (*frame*) või mõnes muus kohas (kettapuhvris)
  - Tühi mälu märgitakse lihtsalt, et "seal kõik nullid"
- Virtuaalsed aadressid ühendatakse reaalsete *frame*ide või selle muu kohaga

Page fault

- Kuna virtuaalmälu on rohkem kui päris mälu, tuleb *page*sid pärismälu ja tagavaramälu vahel vahetada
- Tegevus nimega *paging*
- *Page fault*: katse lugeda *page*, millel puudub vaste reaalses mälus
- Kui toimub, loetakse leht tagavaramälust "päris" mällu: paari millisekundine operatsioon võtab järsku suurusjärgu võrra kauem aega
- Kui "ketas ragistab", on palju *page fault*e

Segmendid

- Töötav programm paikneb mälus kolmes segmendis:
  - *text segment*: programmitekst e. käivitatav kood.
  - *data segment*: programmi andmete segment, exec hõivab seda ette, saab ise juurde võtta-
  - *stack segment*: programmi *stack*; kasvab vajadusel, ei kahane. (Ehk siis: segment ei kahane, *stack* ise võib kahaneda.)

Viisid mälu saamiseks

- Mälu saamiseks on kaks viisi
  - käivitades (exec) kui programm mällu loetakse eraldatatakse mälu programmitekstile, konstantidele ja staatilistena deklareeritud muutujatele
  - programselt:
    - automaatsed muutujad
    - malloc
    - mmap: faili virtuaalse mäluga vastavusse seadmine
  - fork: *copy on write trikk*
- Programmi lõpetamisel mälu eraldi ei vabastata: kogu aadressruum kaob ära

Staatilised ja Automaatsed muutujad

- Staatiline hõivamine: Globaalmuutujad ja *static* tüüpi andmed. Hõivatakse programmi käivitumisel ja ei vabastata kunagi
- Automaatne hõivamine: Juhtub siis, kui deklareerida automaatne muutuja nagu funktsiooniargument või kohalik muutuja. Hõivatakse mälu deklareerivasse blokki sisenedes ning vabastatakse sealt väljumisel

- Võimalik C keele enda abil

Dünaamiline hõivamine

- Librade abil saab ise mälu juurde küsida.
  - malloc(), realloc(), calloc()
- Ühtki muutujat ei saa kunagi hoida dünaamilises mälus (seepärast kasutame pointereid)
- GNU malloc
  - ei fragmenteeri mälu (kõrvalseisvad vabad tükid ühendatakse probleemideta)
  - väga suured (kõvasti suuremad kui *page*) hõivatakse mmap() abil:
    - läve määramine funktsiooniga mallopt()

Mäluhalduse jälgimine

- Funktsioon ja lõbusa nimega funktsioon:

```
void mtrace(void);
void muntrace(void);
```

- Salvestavad keskkonnamuutujaga MALLOC_TRACE määratud nimega faili mälukasutusstatistika mälu hõivamiste ja vabastamiste kohta.
- Esimene aktiveerib, teine deaktiveerib seire
- GNU spetsiifiline: mcheck.h failist saab
- Fail ei ole inimloetav. Arusaamiseks programm:

```
mtrace programminimi mtrace-log
```

mmap()

- mmap() poogib faili loetavasse virtuaalmällu (või teeb seda anonüümselt)
- Kohati tõhusam:
  - Loeme mällu ainult jupid, mida ka reaalselt kasutame
  - mmap() asjad saab vajadusel kettale tagasi kirjutada
  - saame avada faile, mis on suuremad kui mem+swap

```
void * mmap (void *address, size_t length, int protect,
             int flags, int filedes, off_t offset)
```

  - Parameetrid: aadress kuhu soovime mappingut, pikkus, kuidas kaitsta, kuidas hallata, failideskriptor ja millisest faili punktist alustada

mmap() parameetritest

- prot: PROT_READ, PROT_WRITE, PROT_EXEC bitid
  - sõltuvalt süsteemist võib esineda anomaaliaid: *write* on enamasti ka *read* või kirjutuskaitstud failidesse ei saa kirjutada ka siis, kui PROT_READ puudub
- flags: *mappingu* olemusest:
  - MAP_PRIVATE: faili tagasi ei kirjutata, muutmisel kirjutatakse tavamällu ja hallatakse sõltumatult
  - MAP_SHARED: muutused kajastuvad ka failis & teiste protsesside jaoks
  - MAP_FIXED: nõua kindel aadress või ebaõnnestu
  - MAP_ANONYMOUS: ära seo failiga (mõni süsteem küsib täpselt nii heap-i juurde)

munmap() & msync() & madvise()

- munmap(): Eemaldab *memory mapping*u etteantud aadressist etteantud aadressini (kasvõi mitu tükki korraga); võib sisaldada ka *mapping*uta lõike.
- msync(): Kirjutab *mapping*u etteantud aadressist etteantud mahus faili.
- madvise(): selgitab kernelile aadressivahemiku mappingu iseloomu: kas kasutame juhupöördumist, loeme järjest, läheb igal juhul kõike vaja, või ei vaja enam üldse ja ta võib täiesti kõik seal sees ära unustada ja klient ka ei lahku toast hüsteeriliselt karvu katkudes

Long Jump

```
#include <setjmp.h>
int setjmp(jmp_buf env);
void longjmp(jmp_buf env, int val);
```

- setjmp ja longjmp aitavad katkestuste haldamise ajal programmi teises kohas jätkama panna.
- setjmp() salvestab *stack*i sisu/keskkonna env puhvrisse, et seda hiljem longjmp() väljakutsel kasutada. Kui setjmp() välja kutsunud funktsioon tagastub, kaotab env sisu oma mõtte.
- setjmp() tagastab 0, kui tagastub esimest korda ja nullist erineva väärtuse, kui tagastub pärast longjmp() väljakutset

Long Jump (2)

```
void longjmp(jmp_buf env, int val);
```

- longjmp() taastab keskkonna, milles setjmp() välja kutsuti. Pärast longjmp() väljakutset käitub programm justkui setjmp() oleks tagastanud väärtuse val. longjmp() abil ei ole võimalik 0 väärtust saata: see asendub väärtusega 1

```
jmp_buf env;
if ((val=setjmp(env)) == 0)
   printf("Now we have set long jump\n");
else
   printf("Long jump has returned value
%d\n",val);
..........…
longjump(env, 3);
```

long.c

Teretulemast loengu lõppu

Slaide selles loengus: 17